**12 April 2000** **Demmel / Shewchuk**

# CS 170: Midterm Exam II

University of California at Berkeley
Department of Electrical Engineering and Computer Sciences
Computer Science Division

This is a closed book, closed calculator, closed computer, closed network, open brain exam, but you are permitted a one-page, double-sided set of notes.

Write all your answers on this exam. If you need scratch paper, ask for it, write your name on each sheet, and attach it when you turn it in (we have a stapler).

**Do not open your exam until you are told to do so!**

(1 point) **Name:**_____

(1 point) **Login:**_____

(1 point) **Lab TA:**_____

(1 point) **Neighbor to your left:**_____

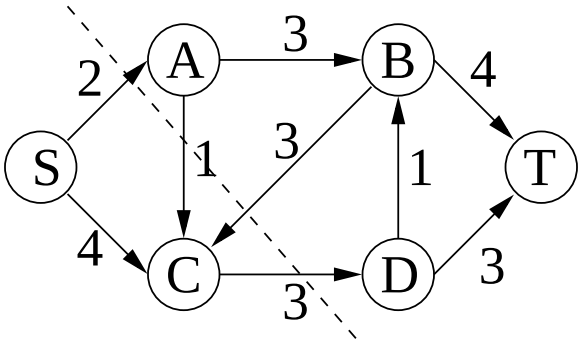
(1 point) **Neighbor to your right:**_____

*Do not write in these boxes.*

| **Problem #** | **Possible** | **Score** |
|---|---|---|
| Name, etc. | 5 | |
| 1. Network Flows | 12 | |
| 2. True/False | 16 | |
| 3. NP-Completeness | 12 | |
| 4. Dynamic Programming | 15 | |
| Total | 60 | |

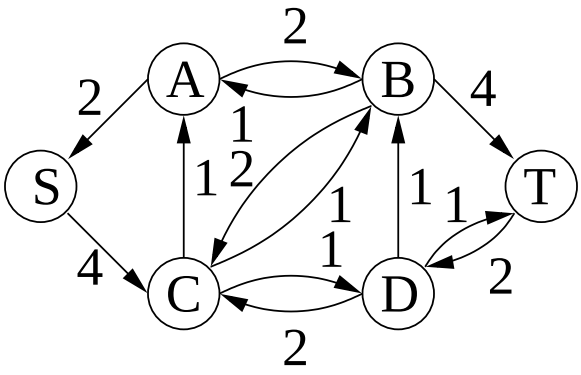
**Problem 1.** (12 points) **Network Flows**

In the flow network illustrated below, each directed edge is labeled with its capacity. We are using the Ford-Fulkerson algorithm to find the maximum flow. The first augmenting path is S-A-C-D-T, and the second augmenting path is S-A-B-C-D-T.




a. (6 points) Draw the residual network after we have updated the flow using these two augmenting paths (in the order given).

**ANSWER.**




b. (4 points) List all of the augmenting paths that could be chosen for the third augmentation step.

**ANSWER.**

S-C-A-B-T, S-C-B-T, S-C-D-B-T, and S-C-D-T.

c. (2 points) What is the numerical value of the maximum flow? Draw a dotted line through the original graph to represent the minimum cut.

**ANSWER.**

5. See top figure above for the minimum cut.

**Problem 2.** (16 points) **True or False?**

Each true/false question is worth two points. You will **lose** two points for a wrong answer, so only answer if you think you are likely to be right.

a. $\log(c\sqrt{n}) \in O(\log \sqrt{n})$ for all $c > 0$.

**ANSWER.**

True. Observe that $\log(c\sqrt{n}) = \log c + \log \sqrt{n}$.

b. $\log^* \log n \in \Theta(\log^* n)$.

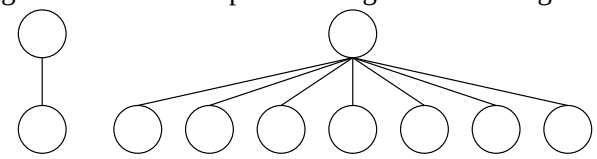
**ANSWER.**

True. Observe that $\log^* \log n = (\log^* n) - 1$. (Think about the definition of $\log^*$ to see why.)

c. A *grandchild* in a forest-based disjoint-set data structure is any node that has a grandparent. (In other words, neither the node nor its parent is a root.) If we use union-by-rank, a single union operation cannot increase the total number of grandchildren by more than $n/2$, where $n$ is the total number of nodes.

**ANSWER.**

False. The number of grandchildren can increase by up to $n - 3$. Consider the following example, with both roots having rank 1. A union operation might make the right root a child of the left.



d. Let $f$ be a non-empty file that uses 3-bit fixed-length codewords for the characters a–h. The file cannot use any character other than these eight. True or false: Huffman coding can always compress any such file (i.e., produce an encoding with fewer bits). (Don't count the bits needed to express the mapping from bits to alphabet.)

**ANSWER.**

False. For example, if all eight characters occur with the same frequency, the Huffman encoding will assign each character a 3-bit codeword.

e. If A is an NP-complete problem, and problem B reduces (polynomially) to A, then B is an NP-complete problem.

**ANSWER.**

False. B might be much easier. For example, the problem of determining whether a string of bits includes a 1 can be easily reduced to SAT, but it's certainly not an NP-hard problem.

f. The problem of finding the length of the shortest path between two nodes in a directed graph (which we would normally solve using Dijkstra's algorithm) can be expressed as a linear program. (An ordinary, continuous linear program; we're not talking about integer linear programs.) Assume that there is only one shortest path.

**ANSWER.**

True. For example, we can formulate this as a min-cost flow problem. (Note that I didn't say *max*-flow.) Assign every edge of the graph a capacity of 1, and a cost equal to its weight. We treat this

flow problem as a linear program with the usual capacity constraints (the flow through any edge is not less than zero and not more than one), the usual conservation constraints (total flow into a node equals total flow out of the node, except for the source and sink), plus the constraint that the amount of flow out of the source is one. The objective function is the usual for min-cost flow: the sum over all edges of the flow through an edge times the cost of the edge. Since there is only one shortest path, the entire one unit of flow will move through the shortest path, and there will be no fractional flows.

g. Consider a linear program with three variables (dimensions). If $x$ is an optimal point of the linear program, then $x$ must be a vertex of the polyhedron formed by the constraints (inequalities).

**ANSWER.**

False. If the objective vector is orthogonal to an edge or a face of the polyhedron, $x$ might lie anywhere within that edge or face.

h. Imagine we have a computer with an infinite amount of RAM, in which any memory address can be accessed in constant time. There exists a computer program of infinite length that solves SAT in linear time.

**ANSWER.**

True. The simplest approach is to have an infinite table that maps every possible input to the correct answer for that input. Our program merely converts the input into a memory address in the table, and looks up the solution at that address.

If you think it's cheating to have an infinite table of data, we can solve SAT with program code alone. We walk through the input one bit at a time. For each bit, use "if" statements to branch to code for one of three cases: a zero bit, a one bit, or the end of the input string. The branch for the end of a string returns/prints the correct solution for the input string. The other two branches continue by reading the next bit, and by having three-way branches based on that bit.

**Problem 3.** (12 points) **NP-Completeness**

We know that max-flow problems can be solved in polynomial time. However, consider a modified max-flow problem in which every edge must either have zero flow or be completely saturated. In other words, if there is any flow through an edge at all, the flow through the edge is the capacity of the edge. Let's call a flow a *saturated flow* if it satisfies this additional constraint, as well as all the usual constraints of flow problems. The modified problem asks us to find the maximum saturated flow.

a. (1 point) Formulate this problem as a decision (yes/no) problem. (We'll call it MAX-SATURATED-FLOW.) Write your answer in the form of a question.

**ANSWER.**

Is there a saturated flow of value $k$ or greater? (Alternative: is there a saturated flow of value $k$?) Here, $k$ is an arbitrary number.

b. (3 points) Show that MAX-SATURATED-FLOW is in NP.

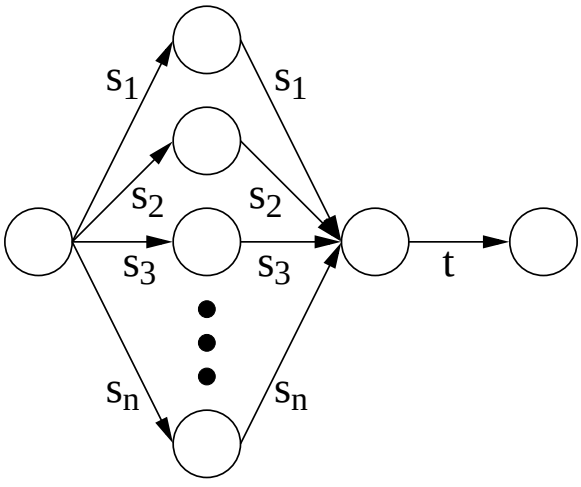
**ANSWER.**

For a certificate, we use a list of all the saturated edges. Given such a list, we can check in linear time the flow out of the source node, and whether the conservation condition for each node (except the source and sink) is satisfied.

c. (8 points) Show that MAX-SATURATED-FLOW is NP-hard. Hint: the easiest reduction is *not* from one of the NP-complete *graph* problems you know. What other NP-complete problems do you know?

**ANSWER.**

It's relatively easy to reduce SUBSET-SUM to MAX-SATURATED-FLOW. Consider the NP-hard problem of finding a subset of $\{s_1, s_2, s_3, \ldots, s_n\}$ that sums to $t$. This problem is solved if we find a saturated flow of value $t$ in the following graph (or alternatively, the maximum saturated flow of this graph).



**Problem 4.** (15 points) **Dynamic Programming**

Consider the *Maximum Alternating Sum Subsequence (MASS)* problem. Given a sequence $S = [x_1, x_2, \ldots, x_n]$ of positive integers, find the subsequence $A = [x_{i_1}, x_{i_2}, \ldots, x_{i_k}]$, where $i_1 < i_2 < \cdots < i_k$, that maximizes the alternating sum

$$x_{i_1} - x_{i_2} + x_{i_3} - x_{i_4} + \cdots \pm x_{i_k}.$$

For example, if $S = [4, 9, 2, 4, 1, 3, 7]$, the MASS is $A = [9, 2, 4, 1, 7]$ which evaluates to $9 - 2 + 4 - 1 + 7 = 17$. If $S = [7, 6, 5, 4, 3, 2, 1]$, the MASS is $A = [7]$. Clearly, the length of the MASS depends on the actual values in $S$. Assume that the length of $S$ is at least one, and all its elements are integers greater than zero.

The following questions ask you to develop a dynamic programming algorithm to find the *value* of the MASS (but not the actual subsequence) for a given sequence. We would like a solution with optimal running time, but you will only lose a few points if you have a correct, suboptimal algorithm.

a. (3 points) Describe the subproblems you will need to solve. How many tables (of subproblem solutions) do you need? What is the dimension of the table(s)? What do the table entries represent? (Hint: it helps to treat subsequences of even and odd lengths separately.)

**ANSWER.**

This question can be answered in several ways; here's one. Let $E$ and $O$ be two one-dimensional tables with indices from zero to $n$. $O[i]$ will store the value of the maximum odd-length alternating sum subsequence of the first $i$ elements of $S$, and $E[i]$ will store the value of the maximum even-length alternating sum subsequence of the first $i$ elements of $S$.

b. (7 points) Write a recursion for the values your algorithm will fill into the table(s). Also write the base case(s); i.e. the table value(s) that bootstrap(s) the recursion.

**ANSWER.**

Recursion: $O(i) = \max\{E(i-1) + x_i, O(i-1)\}$; $E(i) = \max\{O(i-1) - x_i, E(i-1)\}$.
Base cases: $O(0) = -\infty$; $E(0) = 0$.

($O(0) = 0$ is acceptable, but note that it wouldn't work if we allowed $S$ to contain negative integers.)

c. (4 points) Write iterative, non-recursive pseudocode for your algorithm.

**ANSWER.**

$O[0] \leftarrow -\infty$
$E[0] \leftarrow 0$
for $i \leftarrow 1$ to $n$
    $O[i] = \max\{E[i-1] + x_i, O[i-1]\}$
    $E[i] = \max\{O[i-1] - x_i, E[i-1]\}$
return $O[n]$

If $S$ could contain negative integers, the last line would have to return $\max\{O[n], E[n]\}$.

d. (1 point) What is the running time of your algorithm?

**ANSWER.**

Ours runs in $\Theta(n)$ time.